A Logic for Algebraic Effects

A Logic for Algebraic Effects

Gordon Plotkin [*] Matija Pretnar [†]
gdp@inf.ed.ac.uk matija@pretnar.info

Laboratory for Foundations of Computer Science,
School of Informatics, University of Edinburgh,
Edinburgh, Scotland

Abstract

We present a logic for algebraic effects, based on the algebraic representation of computational effects by operations and equations. We begin with the a-calculus, a minimal calculus which separates values, effects, and computations and thereby canonises the order of evaluation. This is extended to obtain the logic, which is a classical first-order multi-sorted logic with higher-order value and computation types, as in Levy's call-by-push-value, a principle of induction over computations, a free algebra principle, and predicate fixed points. This logic embraces Moggi's computational λ-calculus, and also, via definable modalities, Hennessy-Milner logic, and evaluation logic, though Hoare logic presents difficulties.

1 Introduction

Numerous approaches have sprung up to tackle the complexity of reasoning about programming languages that incorporate computational effects such as exceptions, nondeterminism, state, input/output, concurrency, or continuations.

Moggi gave a uniform representation of effects by monads [14], with the idea that computations for an element of (say) a set A are modelled by elements of TA, where T is the monad. Plotkin and Power then proposed representing the effects by operations and equations [19, 21, 23] to get a uniform theory of effects that accounted for their source: we call such effects *algebraic*. All of the effects mentioned above are algebraic, with the notable exception of continuations [3], which have to be treated differently [9].

In the algebraic approach, the arguments of an operation represent possible computations after the occurrence of an effect. For example, using a binary choice operation or, a nondeterministically chosen boolean is represented by the term $\mathsf{or}(\mathsf{return}\,\mathsf{true}, \mathsf{return}\,\mathsf{false})$; the same operation can be used for a choice between two elements of any given type. The equations for the operations, for example saying that or is a semi-lattice operation, generate a free algebra monad, which is exactly the monad proposed by Moggi [20] to model the corresponding effect.

This article proposes a logic for algebraic effects [22], and aims to show that it provides a rich framework, which embraces both approaches that have developed around specific effects, such as Hennessy-Milner logic [7] for concurrency, and more abstract approaches originating from the representation of computational effects with monads, such as Pitts' evaluation logic [17, 15, 16]. (We define an *embrace* to be a translation, which preserves provable judgements. If the translation also reflects provable judgements, we call it a *strong embrace*.)

Section 2 introduces the a-calculus, its syntactic properties, and its denotational semantics. The a-calculus is a minimal calculus which separates values, effects, and computations, thereby canonising the order of evaluation. In Section 3 it is extended to a classical first-order multi-sorted logic with higher-order value and computation types, as in Levy's call-by-push-value [12], a principle of induction over computations, a free algebra principle, and predicate fixed points. Next, in Section 4, we show that Moggi's computational λ-calculus, and, via definable modalities, Hennessy-Milner logic and evaluation logic are all embraced by our logic; we also observe the problems in embracing Hoare logic [8]. In Section 5, we briefly study the introduction of recursion and its logic and semantics. Finally, Section 6 discusses some open problems.

[*]Supported by EPSRC grant GR/586371/01 and a Royal Society-Wolfson Award Fellowship.

[†]Supported by EPSRC grant GR/586371/01.

2 The a-calculus

The a-calculus consists of three parts: one for values, one for effects, and one for computations. This structure is also reflected in the semantics, with each part interpreted in a separate category. This is similar to Levy's call-by-push-value λ-calculus, which consists of a part for values and a part for computations. We first describe values and effects by two equational theories. These serve as parameters to a calculus for computations that use those values and effects.

2.1 Values

We take a collection of *base types* α such as natural numbers $\mathbf{nat}$, booleans $\mathbf{bool}$, or memory locations $\mathbf{loc}$. In the signature Σ_{fun}, we list *base functions* $f : (\alpha_1, \ldots, \alpha_n) \to \beta$, for example $\mathsf{zero} : () \to \mathbf{nat}$, $\mathsf{succ} : (\mathbf{nat}) \to \mathbf{nat}$, or $\mathsf{plus} : (\mathbf{nat}, \mathbf{nat}) \to \mathbf{nat}$.

As shown in Figure 1, we build *value terms* v and type them in a context Γ, consisting of variables x uniquely bound to value types, with typing judgements of the form $\Gamma \vdash v : \sigma$. We write $x : \sigma \in \Gamma$ if x is bound to σ in Γ. Throughout the article, we use vector notation $\vec{a}$ to abbreviate lists $a_1, \ldots, a_n$.

$$\sigma ::= \alpha \qquad v ::= x \mid f(v_1, \ldots, v_n)$$

$$\Gamma \vdash x : \sigma \quad (x : \sigma \in \Gamma)$$

$$\frac{\Gamma \vdash v_i : \alpha_i \quad (i = 1, \ldots, n)}{\Gamma \vdash f(v_1, \ldots, v_n) : \beta} \quad (f : (\vec{\alpha}) \to \beta \in \Sigma_{\text{fun}})$$

Figure 1. Syntax and typing rules for value terms

We describe the properties of values in a *value theory* $\mathfrak{V}$, consisting of equations $\Gamma \vdash v_1 = v_2$ between value terms $\Gamma \vdash v_1 : \sigma$, $\Gamma \vdash v_2 : \sigma$, and closed under the usual rules for multi-sorted equational logic. We write $\Gamma \vdash_{\mathfrak{V}} v_1 = v_2$ if the equation $\Gamma \vdash v_1 = v_2$ is in the value theory $\mathfrak{V}$.

2.2 Effects

To represent the sources of effects, we take a finite single-sorted signature Σ_{op} of finitary algebraic operations $op : n$. Examples are a binary operation $\mathsf{or} : 2$ for nondeterminism, or a family of nullary operations $\mathsf{raise}_e : 0$ with e varying over a finite set E of exceptions.

To capture the polymorphic nature of operations, we build *effect terms*, which serve as templates for computation terms of any given type. Effect terms are built and typed in a context $\Xi = \xi_1, \ldots, \xi_n$ of distinct *effect variables*, as shown in Figure 2. Later, computation terms of an arbitrary type will be substituted for these variables.

$$T ::= \xi \mid op(T_1, \ldots, T_n)$$

$$\Xi \vdash \xi \quad (\xi \in \Xi)$$

$$\frac{\Xi \vdash T_i \quad (i = 1, \ldots, n)}{\Xi \vdash op(T_1, \ldots, T_n)} \quad (op : n \in \Sigma_{\text{op}})$$

Figure 2. Syntax and typing rules for effect terms

An example effect term is $\mathsf{or}(\xi, \mathsf{raise}_e())$, which is an effect term representing a nondeterministic choice between ξ and raising an exception e.

We describe the properties of effects with equations of the form $\Xi \vdash T_1 = T_2$; an *effect theory* $\mathfrak{E}$ is a collection of such equations, closed under the standard rules for equational theories. As for the value theory, we write $\Xi \vdash_{\mathfrak{E}} T_1 = T_2$ if the equation $\Xi \vdash T_1 = T_2$ is in the effect theory $\mathfrak{E}$. Only *equationally consistent* effect theories, that is theories without the equation $\xi_1, \xi_2 \vdash \xi_1 = \xi_2$, are of interest to us.

Some examples of algebraic effects are shown in Table 1, where in the case of state, $\mathsf{lookup}_l(T_1, \ldots, T_n)$ is an effect term that looks up the location l and proceeds as T_i if l contains the datum d_i. In addition to all those effects, we can also represent various combinations of effects [10]. As we demanded that the signature Σ_{op} is finite and the operations are finitary, the sets of exceptions E, locations L, data D, and the alphabet A must all be finite. This restriction will be lifted when we generalise operations in Section 3.1.

2.3 Computations

Effectful programs cause effects, return values, and have an evaluation order. To reflect this, we represent them by *computation terms*, limiting these to: computation terms combined by an operation, returned value terms, and computation terms sequenced with a let binding. And, as seen in Figure 3, we type them with computation types, ranged over by $\underline{\tau}$. In the a-calculus, the computation types are limited to types $F\sigma$ of computations ultimately returning a value of type σ.

We define the *instantiation* $\Gamma \vdash T[\vec{t}/\vec{\xi}] : \underline{\tau}$ of an effect term $\xi_1, \ldots, \xi_n \vdash T$ by computation terms $\Gamma \vdash t_i : \underline{\tau}$, for

effect	operations	equations
a family of exceptions E	$\mathsf{raise}_e : 0$ for each $e \in E$	none
nondeterminism	$\mathsf{or} : 2$	$\mathsf{or}(\xi,\xi) = \xi, \mathsf{or}(\xi_1,\xi_2) = \mathsf{or}(\xi_2,\xi_1),$ $\mathsf{or}(\mathsf{or}(\xi_1,\xi_2),\xi_3) = \mathsf{or}(\xi_1,\mathsf{or}(\xi_2,\xi_3))$
state with locations L, ranging over D	$\mathsf{lookup}_l : \lvert D\rvert$ for each $l \in L$, $\mathsf{update}_{l,d} : 1$ for each $l \in L, d \in D$	seven equational schemas [20]
input/output on an alphabet A	$\mathsf{input} : \lvert A\rvert$, $\mathsf{output}_a : 1$ for each $a \in A$	none

Table 1. Examples of algebraic effects, together with signatures of operations and equations that generate the effect theories

$$\underline{\tau} ::= F\sigma$$
$$t ::= op(t_1,\dots,t_n) \mid \mathsf{return}\, v \mid \mathsf{let}\, x\, \mathsf{be}\, t\, \mathsf{in}\, t'$$

$$\frac{\Gamma \vdash t_i : \underline{\tau} \quad (i = 1,\dots,n)}{\Gamma \vdash op(t_1,\dots,t_n) : \underline{\tau}} \quad (op : n \in \Sigma_{\mathrm{op}})$$

$$\frac{\Gamma \vdash v : \sigma}{\Gamma \vdash \mathsf{return}\, v : F\sigma} \qquad \frac{\Gamma \vdash t : F\sigma \qquad \Gamma, x : \sigma \vdash t' : \underline{\tau}}{\Gamma \vdash \mathsf{let}\, x\, \mathsf{be}\, t\, \mathsf{in}\, t' : \underline{\tau}}$$

Figure 3. Syntax and typing rules for computation terms

$$\frac{\Gamma \vdash_{\mathfrak{V}} v = v'}{\Gamma \vdash \mathsf{return}\, v = \mathsf{return}\, v'}$$

$$\frac{\Xi \vdash_{\mathfrak{E}} T_1 = T_2}{\Gamma \vdash T_1[\vec{t}/\vec{\xi}] = T_2[\vec{t}/\vec{\xi}]}$$

$$\Gamma \vdash \mathsf{let}\, x\, \mathsf{be}\, \mathsf{return}\, v\, \mathsf{in}\, t' = t'[v/x]$$

$$\Gamma \vdash \mathsf{let}\, x\, \mathsf{be}\, op(t_1,\dots,t_n)\, \mathsf{in}\, t' = op(\overrightarrow{\mathsf{let}\, x\, \mathsf{be}\, t\, \mathsf{in}\, t'})$$

Figure 4. Equational logic of the a-calculus

$i = 1,\dots,n$, by

$$\xi_i[\vec{t}/\vec{\xi}] = t_i$$
$$op(T_1,\dots,T_n)[\vec{t}/\vec{\xi}] = op(T_1[\vec{t}/\vec{\xi}],\dots,T_n[\vec{t}/\vec{\xi}])\ .$$

In this way, effect terms yield computation terms of arbitrary type. For example, a computation term $\mathsf{raise}_e()$, which raises an exception e, is of type $\underline{\tau}$ for any computation type $\underline{\tau}$.

The equational logic consists of equations of the form $\Gamma \vdash t_1 = t_2$, where t_1 and t_2 have the same type $\underline{\tau}$ in the context Γ. In addition to the usual congruence rules for equality, the logic has two rules and two equational schemas, given in Figure 4, where we write $op(\overrightarrow{\mathsf{let}\, x\, \mathsf{be}\, t\, \mathsf{in}\, t'})$ as an abbreviation for $op(\mathsf{let}\, x\, \mathsf{be}\, t_1\, \mathsf{in}\, t', \dots, \mathsf{let}\, x\, \mathsf{be}\, t_n\, \mathsf{in}\, t')$. (We use similar abbreviations elsewhere.)

We use the two rules to inherit equations from the value and effect theories. The first equational schema is the usual β-equality for let binding, understanding the second one requires some operational intuition. The evaluation of $\mathsf{let}\, x\, \mathsf{be}\, op(t_1,\dots,t_n)\, \mathsf{in}\, t'$ begins with an occurrence of the effect represented by the operation $op : n$, and then, depending on the outcome of an effect, it proceeds by evaluating one of the computation terms $t_1,\dots,t_n$ and binding its result to x in computation term t': but this is exactly what the schema states.

The separation of values and computations allows us to ensure that each computation term has an explicit evaluation order (cf. administrative[1] normal form [3]) although, admittedly, we give no operational semantics here. This avoids cases where the evaluation order has to be defined by convention, for example which of the t_i is evaluated first in the evaluation of $f(t_1,\dots,t_n)$. It also makes the calculus and its extensions more concise, as each such convention requires an additional axiom, which we rather take as an abbreviation. For example, we regard a computation term of the form $f(t_1,\dots,t_n)$ as abbreviating

$$\mathsf{let}\, x_1\, \mathsf{be}\, t_1\, \mathsf{in} \dots \mathsf{let}\, x_n\, \mathsf{be}\, t_n\, \mathsf{in}\, f(x_1,\dots,x_n)$$

making the usual left-to-right evaluation order explicit.

[1] The letter a in the a-calculus stands for both algebraic and administrative.

The two equational schemas allow us to put each computation term into a canonical form with no let bindings. We use this to derive η-equality and the associativity of let binding, which must usually be taken as axioms [14, 12].

Lemma 1. *For every effect term $\Xi \vdash T$, computation terms $\Gamma \vdash t_i : F\sigma$, for each $\xi_i \in \Xi$, and $\Gamma, x:\sigma \vdash t' : \underline{\tau}$, we have*

$$\Gamma \vdash \mathsf{let}\, x\, \mathsf{be}\, T[\vec{t}/\vec{\xi}]\, \mathsf{in}\, t' = T[\overrightarrow{\mathsf{let}\, x\, \mathsf{be}\, t\, \mathsf{in}\, t'}/\vec{\xi}]\ .$$

Proof. By induction on the structure of T using the commutativity of operations and let binding. □

Definition 2. *A computation term $\Gamma \vdash t : F\sigma$ is in* canonical form*, if it is of the form $T[\overrightarrow{\mathsf{return}\, v}/\vec{\xi}]$ for some effect term $\Xi \vdash T$ and value terms $\Gamma \vdash v_i : \sigma$, for each $\xi_i \in \Xi$.*

Proposition 3. *For every computation term $\Gamma \vdash t : F\sigma$, there exists a computation term $\Gamma \vdash t' : F\sigma$ in canonical form, such that $\Gamma \vdash t = t'$.*

Proof. We proceed by induction on the structure of t. The cases where $t = \mathsf{return}\, v$ or $t = op(t_1, \ldots, t_n)$ are straightforward, while for the case $t = \mathsf{let}\, x\, \mathsf{be}\, t_1\, \mathsf{in}\, t_2$ we use Lemma 1. □

Theorem 4. *The equalities*

$$\Gamma \vdash \mathsf{let}\, x\, \mathsf{be}\, t\, \mathsf{in}\, \mathsf{return}\, x = t$$

and

$$\Gamma \vdash \mathsf{let}\, x_1\, \mathsf{be}\, t_1\, \mathsf{in}(\mathsf{let}\, x_2\, \mathsf{be}\, t_2\, \mathsf{in}\, t) = \mathsf{let}\, x_2\, \mathsf{be}\, (\mathsf{let}\, x_1\, \mathsf{be}\, t_1\, \mathsf{in}\, t_2)\, \mathsf{in}\, t$$

are derivable, where x_1 does not appear free in t.

Proof. In the first equality, t is provably equal to a term of the form $T[\overrightarrow{\mathsf{return}\, v}/\vec{\xi}]$ because of Proposition 3. Hence the equality

$$\Gamma \vdash \mathsf{let}\, x\, \mathsf{be}\, t\, \mathsf{in}\, \mathsf{return}\, x = T[\overrightarrow{\mathsf{let}\, x\, \mathsf{be}\, \mathsf{return}\, v\, \mathsf{in}\, \mathsf{return}\, x}/\vec{\xi}]$$

is derivable by Lemma 1. We finish the proof using β-equality. The proof of the second equality proceeds similarly, assuming now that t_1 is in canonical form. □

As seen in the above proof, the associativity of let binding is a consequence of its commutativity with operations. There are other properties of operations reflected in let binding, for example commutativity is derivable when the effect theory is commutative.

Proposition 5. *If the equality*

$$\Xi \vdash op(op'(\xi_{11}, \ldots, \xi_{1n'}), \ldots, op'(\xi_{n1}, \ldots, \xi_{nn'})) = op'(op(\xi_{11}, \ldots, \xi_{n1}), \ldots, op(\xi_{1n'}, \ldots, \xi_{nn'}))$$

is in the effect theory $\mathfrak{E}$ for all operations $op : n$ and $op' : n'$ in Σ_{op}, then the equality

$$\Gamma \vdash \mathsf{let}\, x_1\, \mathsf{be}\, t_1\, \mathsf{in}\, \mathsf{let}\, x_2\, \mathsf{be}\, t_2\, \mathsf{in}\, t' = \mathsf{let}\, x_2\, \mathsf{be}\, t_2\, \mathsf{in}\, \mathsf{let}\, x_1\, \mathsf{be}\, t_1\, \mathsf{in}\, t'$$

is derivable, assuming x_1 and x_2 are distinct and do not appear free in t_1 and t_2.

If in addition

$$\Xi \vdash op(op(\xi_{11}, \ldots, \xi_{1n}), \ldots, op(\xi_{n1}, \ldots, \xi_{nn})) = op(\xi_{11}, \ldots, \xi_{nn})$$

is in the effect theory $\mathfrak{E}$ for all operations $op : n \in \Sigma_{\mathrm{op}}$, the equality

$$\Gamma \vdash \mathsf{let}\, x_1\, \mathsf{be}\, t\, \mathsf{in}\, \mathsf{let}\, x_2\, \mathsf{be}\, t\, \mathsf{in}\, t' = \mathsf{let}\, x_1\, \mathsf{be}\, t\, \mathsf{in}\, t'[x_1/x_2]$$

is also derivable.

2.4 Semantics

We interpret value terms in the category Set of sets, effect terms in a Lawvere theory L, and computation terms in the category $\mathrm{Mod}_L(\mathsf{Set})$ of models of the theory L in Set.

Values An interpretation $\mathfrak{I}$ is determined by sets $[\![\alpha]\!]$ for each base type α, and functions $[\![f]\!] : [\![\vec{\alpha}]\!] \to [\![\beta]\!]$, where $[\![\vec{\alpha}]\!] = [\![\alpha_1]\!] \times \cdots \times [\![\alpha_n]\!]$, for each base function $f : (\vec{\alpha}) \to \beta$. Unless stated otherwise, we assume a fixed interpretation and omit the index.

Contexts $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$ are interpreted component-wise: $[\![\Gamma]\!] = [\![\sigma_1]\!] \times \cdots \times [\![\sigma_n]\!]$, and the interpretation of value terms is defined inductively by

$$[\![\Gamma \vdash x_i : \sigma_i]\!] = \mathbf{pr}_{\sigma_i}$$
$$[\![\Gamma \vdash f(v_1, \ldots, v_n) : \beta]\!] = [\![f]\!] \circ \langle [\![v_1]\!], \ldots, [\![v_n]\!] \rangle\ .$$

An interpretation $\mathfrak{I}$ is *sound* with respect to the value theory $\mathfrak{V}$, if for each equation $\Gamma \vdash_{\mathfrak{V}} v_1 = v_2$, we have $[\![v_1]\!] = [\![v_2]\!]$. We consider only sound interpretations.

Effects The effect theory $\mathfrak{E}$ gives rise to a Lawvere theory L in a standard way [2, Volume 2, Chapter 3]. Each effect term $\xi_1, \ldots, \xi_m \vdash T$ is interpreted by a morphism $[\![T]\!] : \underline{m} \to \underline{1}$, defined by

$$[\![\Xi \vdash \xi_i]\!] = \mathbf{pr}_i$$
$$[\![\Xi \vdash op(T_1, \ldots, T_n)]\!] = [\![op]\!] \circ \langle [\![T_1]\!], \ldots, [\![T_n]\!] \rangle\ ,$$

where $[\![op]\!]$ is the interpretation of the operation $op : n$ in the Lawvere theory.

Computations A model of a Lawvere theory L in Set is a product preserving functor $M\colon L \to \mathsf{Set}$. Models, together with natural transformations, form a category $\mathrm{Mod}_L(\mathsf{Set})$, which is equipped with a forgetful functor $U\colon \mathrm{Mod}_L(\mathsf{Set}) \to \mathsf{Set}$, which maps a model M to the set $M(\underline{1})$. This functor has a left adjoint F, which takes a set A and constructs the free model FA on it.

Computation types $F\sigma$ are interpreted by free models $F[\![\sigma]\!]$, and computation terms $\Gamma \vdash t : \underline{\tau}$ are interpreted by maps $[\![t]\!]\colon [\![\Gamma]\!] \to U[\![\underline{\tau}]\!]$, defined inductively by

$$\begin{aligned}
[\![\Gamma \vdash op(t_1,\dots,t_n) : \underline{\tau}]\!] &= [\![\underline{\tau}]\!]([\![op]\!]) \circ \langle [\![t_1]\!],\dots,[\![t_n]\!]\rangle \\
[\![\Gamma \vdash \mathsf{return}\, v : F\sigma]\!] &= \eta_{[\![\sigma]\!]} \circ [\![v]\!] \\
[\![\Gamma \vdash \mathsf{let}\, x\, \mathsf{be}\, t\, \mathsf{in}\, t' : \underline{\tau}]\!] &= [\![t']\!]^\dagger \circ \langle \mathbf{id}_\Gamma, [\![t]\!]\rangle\ ,
\end{aligned}$$

where $f^\dagger = U\epsilon \circ UFf \circ \mathbf{st}_{A,B}\colon A \times UFB \to UM$ is the *lifting* of the function $f\colon A \times B \to UM$, and where $\mathbf{st}_{A,B}\colon A \times UFB \to UF(A \times B)$ is the *strength* of the functor UF.

Lemma 6. *For any map $f\colon A \times B \to UM$, and operation $op\colon n$, the diagram below commutes.*

$$\begin{array}{ccc}
A \times (UFB)^n & \xrightarrow{\langle f^\dagger \circ (\mathbf{id}_A \times \mathbf{pr}_i)\rangle_{i=1,\dots,n}} & (UM)^n \\
\downarrow{\scriptstyle \mathbf{id}_A \times FB([\![op]\!])} & & \downarrow{\scriptstyle M([\![op]\!])} \\
A \times UFB & \xrightarrow[f^\dagger]{} & UM
\end{array}$$

Proof. Transposing $f : A \times B \to UM$, we obtain a map $B \to U(M^A)$ and from the adjunction, a model morphism $\hat{f}\colon FB \Rightarrow M^A$. The commutativity of the above diagram then translates to the commutativity of the diagram

$$\begin{array}{ccc}
FB(\underline{n}) & \xrightarrow{\hat{f}_{\underline{n}}} & M^A(\underline{n}) \\
\downarrow{\scriptstyle FB([\![op]\!])} & & \downarrow{\scriptstyle M^A([\![op]\!])} \\
FB(\underline{1}) & \xrightarrow[\hat{f}_{\underline{1}}]{} & M^A(\underline{1})
\end{array}$$

which commutes because of the naturality of $\hat{f}$. □

Proposition 7 (Soundness)**.** *If $\Gamma \vdash t_1 = t_2$ is derivable for computation terms $\Gamma \vdash t_1 : \underline{\tau}$ and $\Gamma \vdash t_2 : \underline{\tau}$, then $[\![t_1]\!] = [\![t_2]\!]$.*

Proof. To show the soundness of equality, we have to go through all the rules of the a-calculus. Proving soundness of the structural rules, the inheritance rules, and β-equality, is straightforward. To show soundness of commutativity between operations and let binding, we use Lemma 6, a known naturality result [21] adapted to a non-monadic setting. □

On a related note, the converse of the rule for inheritance from the value theory is also sound if the effect theory is equationally consistent. We also have a completeness result relative to the value theory, based on Proposition 3 and completeness results for algebraic theories.

Theorem 8 (Completeness)**.** *Let $\Gamma \vdash t_1 : \underline{\tau}$ and $\Gamma \vdash t_2 : \underline{\tau}$ be computation terms. If the equality $[\![t_1]\!] = [\![t_2]\!]$ holds for all sound interpretations $\mathfrak{I}$, the equation $\Gamma \vdash t_1 = t_2$ is derivable.*

3 The logic

To get an expressive framework, we begin with the a-calculus; we then extend the value theory to a first-order logic; we next extend the effect theory with parametric operations with binding, together with equations with side conditions; and we then extend both value and computation terms using an extended type structure, following the pattern of Levy's call-by-push-value [12].

Finally, we extend the equational logic of the a-calculus to a classical multi-sorted first-order sequent calculus with a principle of induction over computations and predicate fixed points. The terms of this logic are value and computation terms, so according to Pnuelli's classification [25], our logic is an exogenous logic, as computation terms are parts of propositions, rather than an endogenous logic, where all propositions concern a single computation.

3.1 Syntax

First-order value theory As before, we have a collection of base types α and a signature Σ_{fun}, consisting of base functions $f : (\vec{\alpha}) \to \beta$; we also have a signature Σ_{rel} consisting of *base relations* $R : (\vec{\alpha}) \to \mathbf{form}$. We build first-order multi-sorted *value formulae* $\Gamma \vdash \varphi : \mathbf{form}$ in the usual way. A *value theory* $\mathfrak{V}$ is a collection of such formulae, closed under the standard rules for classical multi-sorted first-order logic over the signatures Σ_{fun} and Σ_{rel}.

Parametric operations with binding Instead of having a set of nearly identical operations such as $\mathsf{update}_{l,d} : 1$ for each location l and datum d, we take a single operation with parameter types such as $\mathsf{update} : \mathbf{loc}, \mathbf{dat}; 1$. In this way, we get a finitary syntax describing an infinite set of effects.

Next, if we were to describe a memory holding an infinite set of data by routinely generalising the operations to infinitary ones, we would be left with an infinitary syntax [19]. We take an alternative approach [18], and allow each argument of an operation to be dependent on values of base types, for example $\mathsf{lookup}_l((d : \mathbf{dat}).\mathsf{update}_{l',d}(\xi))$ is an effect term for a computation that copies the datum d from l to l' and proceeds as ξ, using an operation $\mathsf{lookup} : \mathbf{loc}; \mathbf{dat}$.

So we take a more general signature Σ_{op} with operations $op : \vec{\beta}; \vec{\alpha}_1, \ldots, \vec{\alpha}_n$ where the base types $\vec{\beta}$ are the *parameter*, or the *coarity*, types, and the base types $\vec{\alpha}_1, \ldots, \vec{\alpha}_n$ are the respective *arity* types. When writing signatures, we omit the semicolon in $\vec{\beta}; \vec{\alpha}_1, \ldots, \vec{\alpha}_n$ when $\vec{\beta}$ is empty, and we write n instead of $\vec{\alpha}_1, \ldots, \vec{\alpha}_n$ when all the $\vec{\alpha}_i$ are empty.

To reflect the dependency on values, we type effect terms as $\Gamma; \Xi \vdash T$ in a context Γ of value variables $x : \alpha$ and a context Ξ of abstracted effect variables $\xi : (\vec{\alpha})$, according to the following rules, where $op : \vec{\beta}; \vec{\alpha}_1, \ldots, \vec{\alpha}_n$.

$$\frac{\Gamma \vdash \vec{v} : \vec{\alpha}}{\Gamma; \Xi \vdash \xi(\vec{v})} \quad (\xi : (\vec{\alpha}) \in \Xi)$$

$$\frac{\Gamma \vdash \vec{v} : \vec{\beta} \qquad \Gamma, \vec{x}_i : \vec{\alpha}_i; \Xi \vdash T_i \quad (i = 1, \ldots, n)}{\Gamma; \Xi \vdash op_{\vec{v}}((\vec{x}_1 : \vec{\alpha}_1).T_1, \ldots, (\vec{x}_n : \vec{\alpha}_n).T_n)} .$$

To describe the case when an equation holds only for a particular subset of parameters, we write equations of the form $\Gamma; \Xi \vdash T_1 = T_2 \ (\varphi)$, where $\Gamma \vdash \varphi : \mathbf{form}$ is a side condition. In this way, we can use a finite syntax to write down a possibly infinite number of equations. An *effect theory* $\mathfrak{E}$ is a finite collection of such equations, rather than an equational theory. (Unfortunately, we do not know what the rules for equational theories should be when operations have parameters and arguments with binding; see [18] for preliminary results.)

Value and computation terms The types of the calculus are given by

$$\begin{aligned} \sigma &::= \alpha \mid \mathbf{1} \mid \sigma_1 \times \sigma_2 \mid \mathbf{0} \mid \sigma_1 + \sigma_2 \mid U\underline{\tau} \\ \underline{\tau} &::= F\sigma \mid \mathbf{1} \mid \underline{\tau}_1 \times \underline{\tau}_2 \mid \sigma \to \underline{\tau} \, , \end{aligned}$$

while the terms are given by

$$\begin{aligned} v ::= {} & x \mid f(\vec{v}) \mid \star \mid \langle v_1, v_2 \rangle \mid \mathsf{fst}\, v \mid \mathsf{snd}\, v \mid \mathsf{in_0}\, v \mid \mathsf{inl}\, v \mid \mathsf{inr}\, v \mid \\ & \mathsf{match}\, v \,\mathsf{with}\, \mathsf{inl}\, x_1 : \sigma_1 . t_1, \mathsf{inr}\, x_2 : \sigma_2 . t_2 \mid \mathsf{thunk}\, t \\ t ::= {} & \zeta \mid op_{\vec{v}}((\vec{x}_1 : \vec{\alpha}_1).t_1, \ldots, (\vec{x}_n : \vec{\alpha}_n).t_n) \mid \mathsf{force}\, v \mid \\ & \mathsf{return}\, v \mid \mathsf{let}\, x \,\mathsf{be}\, t \,\mathsf{in}\, t' \mid \star \mid \langle t_1, t_2 \rangle \mid \mathsf{fst}\, t \mid \mathsf{snd}\, t \mid \\ & \lambda x : \sigma . t \mid tv \, . \end{aligned}$$

With thunking and forcing, value and computation terms become intertwined: we can thunk each computation term to obtain a value term, which we pass around before eventually forcing it to retrieve the original computation term.

We type value and computation terms in a context Γ of value variables $x : \sigma$ and a context Δ of computation variables $\zeta : \underline{\tau}$. Omitting the previously mentioned rules for base functions, returned values, and let binding, and the well-known rules for variables, products, sums, and function types, the typing rules are:

$$\frac{\Gamma; \Delta \vdash t : \underline{\tau}}{\Gamma; \Delta \vdash \mathsf{thunk}\, t : U\underline{\tau}} \qquad \frac{\Gamma; \Delta \vdash v : U\underline{\tau}}{\Gamma; \Delta \vdash \mathsf{force}\, v : \underline{\tau}}$$

$$\frac{\Gamma; \Delta \vdash \vec{v} : \vec{\beta} \qquad \Gamma, \vec{x}_i : \vec{\alpha}_i; \Delta \vdash t_i : \underline{\tau} \quad (i = 1, \ldots, n)}{\Gamma; \Delta \vdash op_{\vec{v}}((\vec{x}_1 : \vec{\alpha}_1).t_1, \ldots, (\vec{x}_n : \vec{\alpha}_n).t_n) : \underline{\tau}}$$

We define an *instantiation* $\Gamma; \Delta \vdash T[\overrightarrow{(\vec{x} : \vec{\alpha}).t}/\vec{\xi}] : \underline{\tau}$ of an effect term $\Gamma; \Xi \vdash T$ by $\Gamma, \vec{x}_i : \vec{\alpha}_i; \Delta \vdash t_i : \underline{\tau}$, for each $\xi_i : (\vec{\alpha}_i) \in \Xi$. It is defined argument-wise for operations by

$$op_{\vec{v}}(\overrightarrow{(\vec{x}' : \vec{\alpha}').t'})[\overrightarrow{(\vec{x} : \vec{\alpha}).t}/\vec{\xi}] = op_{\vec{v}}(\overrightarrow{(\vec{x}' : \vec{\alpha}').t'[\overrightarrow{(\vec{x} : \vec{\alpha}).t}/\vec{\xi}]})$$

and for variables by

$$\xi_i(\vec{v})[\overrightarrow{(\vec{x} : \vec{\alpha}).t}/\vec{\xi}] = t_i[\vec{v}/\vec{x}_i] \, .$$

(We do not propose any calculus for value and computation terms. It would be natural, for example, to consider conditional equations of the form $\Gamma; \Delta \vdash t = t' \ (\varphi)$, but the difficulty would again be to find the right rules.)

3.2 Logic

As noted before, our logic is an exogenous one, so to describe properties of computations, we introduce predicates π and predicate variables X in addition to the usual propositions φ, all built as:

$$\begin{aligned} \varphi ::= {} & v_1 = v_2 \mid t_1 = t_2 \mid R(\vec{v}) \mid \pi(\vec{v}; \vec{t}) \mid \\ & \bot \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x : \sigma . \varphi \mid \exists \zeta : \underline{\tau} . \varphi \\ \pi ::= {} & X \mid (\vec{x} : \vec{\sigma}; \vec{\zeta} : \vec{\underline{\tau}}).\varphi \mid \mu X : (\vec{\sigma}; \vec{\underline{\tau}}).\pi \end{aligned}$$

where, in $\mu X : (\vec{\sigma}; \vec{\underline{\tau}}).\pi$, the predicate variable X is required to occur positively in π.

We type propositions $\Gamma; \Delta; \Pi \vdash \varphi : \mathbf{prop}$ and predicates $\Gamma; \Delta; \Pi \vdash \pi : (\vec{\sigma}; \vec{\underline{\tau}}) \to \mathbf{prop}$ in a context Γ of value variables $x : \sigma$, a context Δ of computation variables $\zeta : \underline{\tau}$, and a context Π of predicate variables $X : (\vec{\sigma}; \vec{\underline{\tau}}) \to \mathbf{prop}$, according to

$$\frac{\Gamma, \vec{x} : \vec{\sigma}; \Delta, \vec{\zeta} : \vec{\underline{\tau}}; \Pi \vdash \varphi : \mathbf{prop}}{\Gamma; \Delta; \Pi \vdash (\vec{x} : \vec{\sigma}; \vec{\zeta} : \vec{\underline{\tau}}).\varphi : (\vec{\sigma}; \vec{\underline{\tau}}) \to \mathbf{prop}}$$

$$\frac{\Gamma; \Delta; \Pi, X : (\vec{\sigma}; \vec{\underline{\tau}}) \to \mathbf{prop} \vdash \pi : (\vec{\sigma}; \vec{\underline{\tau}}) \to \mathbf{prop}}{\Gamma; \Delta; \Pi \vdash \mu X : (\vec{\sigma}; \vec{\underline{\tau}}).\pi : (\vec{\sigma}; \vec{\underline{\tau}}) \to \mathbf{prop}}$$

and other standard rules.

The judgements of the logic are of the form

$$\Gamma; \Delta; \Pi \mid \Psi \vdash \varphi \ ,$$

where the list of *hypotheses* $\Psi = \varphi_1, \ldots, \varphi_n$ and the *conclusion* φ are all propositions in the contexts $\Gamma; \Delta; \Pi$. We write $\Gamma; \Delta; \Pi \vdash \varphi$ when there are no hypotheses.

The rules of the logic are: standard reasoning rules for a classical first-order sequent calculus, including structural rules; an equivalence

$$\Gamma, \vec{x}\!:\!\vec{\sigma}; \Delta, \vec{\zeta}\!:\!\vec{\underline{\tau}}; \Pi \vdash ((\vec{x}\!:\!\vec{\sigma}; \vec{\zeta}\!:\!\vec{\underline{\tau}}).\varphi)(\vec{x}; \vec{\zeta}) \Leftrightarrow \varphi$$

defining the behaviour of predicates; rules for sums, products, lambda expressions, thunking, and forcing, all as in call-by-push-value [12]; rules

$$\frac{\Gamma \vdash_{\mathfrak{V}} \varphi}{\Gamma; \Delta; \Pi \vdash \varphi}$$

$$\frac{\Gamma; \Xi \vdash_{\mathfrak{E}} T_1 = T_2 \ (\varphi)}{\Gamma; \Delta; \Pi \mid \varphi \vdash T_1[\overrightarrow{(\vec{x}\!:\!\vec{\alpha}).t}/\vec{\xi}] = T_2[\overrightarrow{(\vec{x}\!:\!\vec{\alpha}).t}/\vec{\xi}]}$$

for inheriting from the value and effect theories; two equations

$$y\!:\!\sigma; \zeta\!:\!\sigma \to \underline{\tau} \vdash \mathsf{let}\, x\, \mathsf{be}\, \mathsf{return}\, y\, \mathsf{in}\, \zeta x = \zeta y$$

$$\begin{aligned} \vec{y}\!:\!\vec{\beta}; \vec{\zeta}\!:\!\overrightarrow{\textstyle\prod \vec{\alpha} \to F\sigma}, \zeta'\!:\!\sigma \to \underline{\tau} \vdash \\ \mathsf{let}\, x\, \mathsf{be}\, op_{\vec{y}}(\overrightarrow{(\vec{x}\!:\!\vec{\alpha}).\zeta\langle\vec{x}\rangle})\, \mathsf{in}\, \zeta' x \\ = op_{\vec{y}}(\overrightarrow{(\vec{x}\!:\!\vec{\alpha}).\, \mathsf{let}\, x\, \mathsf{be}\, \zeta\langle\vec{x}\rangle\, \mathsf{in}\, \zeta' x}) \end{aligned}$$

about let binding, the second one for all $op\!:\!\vec{\beta}; \vec{\alpha}_1, \ldots, \vec{\alpha}_n$; an equation

$$\begin{aligned} \vec{y}\!:\!\vec{\beta}; \vec{\zeta}\!:\!\overrightarrow{\textstyle\prod \vec{\alpha} \to \underline{\tau}}, \vec{\zeta}'\!:\!\overrightarrow{\textstyle\prod \vec{\alpha} \to \underline{\tau}'} \vdash \\ op_{\vec{y}}(\overrightarrow{(\vec{x}\!:\!\vec{\alpha}).\langle\zeta\langle\vec{x}\rangle, \zeta'\langle\vec{x}\rangle\rangle}) \\ = \langle op_{\vec{y}}(\overrightarrow{(\vec{x}\!:\!\vec{\alpha}).\zeta\langle\vec{x}\rangle}), op_{\vec{y}}(\overrightarrow{(\vec{x}\!:\!\vec{\alpha}).\zeta'\langle\vec{x}\rangle})\rangle \ , \end{aligned}$$

that defines the behaviour of an operation $op\!:\!\vec{\beta}; \vec{\alpha}_1, \ldots, \vec{\alpha}_n$ on a computation type $\underline{\tau} \times \underline{\tau}'$, and two similar ones for computation types $\mathbf{1}$ and $\sigma \to \underline{\tau}$; two rules stating that the predicate $\mu X\!:\!(\vec{\sigma}; \vec{\underline{\tau}}).\pi$ is the smallest pre-fixed point

$$\frac{\Gamma, \vec{x}\!:\!\vec{\sigma}; \Delta, \vec{\zeta}\!:\!\vec{\underline{\tau}}; \Pi \mid \Psi, \pi[\pi'/X](\vec{x}; \vec{\zeta}) \vdash \pi'(\vec{x}; \vec{\zeta})}{\Gamma, \vec{x}\!:\!\vec{\sigma}; \Delta, \vec{\zeta}\!:\!\vec{\underline{\tau}}; \Pi \mid \Psi, (\mu X\!:\!(\vec{\sigma}; \vec{\underline{\tau}}).\pi)(\vec{x}; \vec{\zeta}) \vdash \pi'(\vec{x}; \vec{\zeta})} \ ;$$

a *principle of induction over computations*, stating that every computation term of type $F\sigma$ is either a returned value, or built from other computation terms using operations, which for a computation variable ζ in a proposition $\Gamma; \Delta, \zeta\!:\!F\sigma; \Pi \vdash \varphi\!:\!\mathbf{prop}$ is of the form

$$\Gamma; \Delta; \Pi \mid \forall x\!:\!\sigma.\varphi[\mathsf{return}\, x/\zeta], \varphi_{op_1}, \ldots, \varphi_{op_k} \vdash \forall \zeta\!:\!F\sigma.\varphi$$

where $op_1, \ldots, op_k$ are all the operations in Σ_{op}, and for $op\!:\!\vec{\beta}; \vec{\alpha}_1, \ldots, \vec{\alpha}_n \in \Sigma_{\mathrm{op}}$, proposition φ_{op} is

$$\begin{aligned} \forall \vec{\zeta}'\!:\!\overrightarrow{\textstyle\prod \vec{\alpha} \to F\sigma}.(\bigwedge_{i=1}^{n} (\forall \vec{x}_i\!:\!\vec{\alpha}_i.\varphi[\zeta_i'\langle\vec{x}_i\rangle/\zeta]) \\ \Rightarrow \forall \vec{y}\!:\!\vec{\beta}.\varphi[op_{\vec{y}}(\overrightarrow{(\vec{x}\!:\!\vec{\alpha}).\zeta'\langle\vec{x}\rangle})/\zeta]) \ ; \end{aligned}$$

and a *free algebra principle*, stating

$$\begin{aligned} \Gamma; \Delta; \Pi \mid \varphi_1, \ldots, \varphi_m \vdash \forall \zeta\!:\!\sigma \to \sigma'.\exists! \zeta^\dagger\!:\!UF\sigma \to \sigma'. \\ (\forall x\!:\!\sigma.\zeta^\dagger(\mathsf{thunk}\, \mathsf{return}\, x) = \zeta(x)) \wedge \psi_{op_1} \wedge \cdots \wedge \psi_{op_k} \ , \end{aligned}$$

where for each $op\!:\!\vec{\beta}; \vec{\alpha}_1, \ldots, \vec{\alpha}_n \in \Sigma_{\mathrm{op}}$, we take a computation term $\vec{y}\!:\!\vec{\beta}; \vec{\zeta}\!:\!\overrightarrow{\prod \vec{\alpha} \to F\sigma'} \vdash t_{op}\!:\!F\sigma'$, and where ψ_{op} is

$$\begin{aligned} \forall \vec{y}\!:\!\vec{\beta}; \vec{\zeta}'\!:\!\overrightarrow{\textstyle\prod \vec{\alpha} \to F\sigma}.\zeta^\dagger\, \mathsf{thunk}\, op_{\vec{y}}(\overrightarrow{(\vec{x}\!:\!\vec{\alpha}).\vec{\zeta}'\langle\vec{x}\rangle}) \\ = t_{op}[\overrightarrow{\lambda\langle\vec{x}\rangle\!:\!\textstyle\prod \vec{\alpha}.\zeta^\dagger\, \mathsf{thunk}\, \zeta'\langle\vec{x}\rangle}/\vec{\zeta}] \ , \end{aligned}$$

and where φ_i states

$$\forall \vec{y}'\!:\!\vec{\beta}', \vec{\zeta}'\!:\!\overrightarrow{\textstyle\prod \vec{\alpha}' \to F\sigma'}.\varphi_i' \Rightarrow T_i[\vec{t}_{op}/\vec{op}] = T_i'[\vec{t}_{op}/\vec{op}]$$

for each equality $\vec{y}'\!:\!\vec{\beta}'; \vec{\xi}\!:\!\overrightarrow{(\vec{\alpha}')} \vdash T_i = T_i' \ (\varphi_i')$ in the effect theory $\mathfrak{E}$, and where $T[\vec{t}_{op}/\vec{op}]$ is defined by

$$\begin{aligned} \xi_j(\vec{v})[\vec{t}_{op}/\vec{op}] = \zeta_j'\langle\vec{v}\rangle \\ op_{\vec{v}}(\overrightarrow{(\vec{x}\!:\!\vec{\alpha}).T}) = t_{op}[\vec{v}/\vec{y}, \overrightarrow{\lambda\langle\vec{x}\rangle\!:\!\textstyle\prod \vec{\alpha}.T[\vec{t}_{op}/\vec{op}]}/\vec{\zeta}] \end{aligned}$$

and where $\forall \zeta\!:\!\sigma \to \sigma'.\varphi$ abbreviates

$$\forall \zeta\!:\!\sigma \to F\sigma'.(\forall x\!:\!\sigma.\exists y\!:\!\sigma'.\zeta x = \mathsf{return}\, y) \Rightarrow \varphi$$

and similarly for existential quantification. (Note that the uniqueness of $\zeta^\dagger$ can be proved using the induction principle.)

In the free algebra principle, t_{op} defines op on σ', formula ϕ_i says that the i^{th} axiom in the effect theory holds in σ', and ψ_{op} says that $\zeta^\dagger$ preserves op. Note that the finiteness of both the signature Σ_{op} and the effect theory $\mathfrak{E}$ are used in the formulation of the induction and free algebra principles.

With the logic presented, we can prove a stronger, non-schematic, version of Theorem 4.

Theorem 9. *The equalities*

$$\zeta\!:\!F\sigma \vdash \mathsf{let}\, x\, \mathsf{be}\, \zeta\, \mathsf{in}\, \mathsf{return}\, x = \zeta$$

and

$$\begin{aligned} \zeta_1\!:\!F\sigma_1, \zeta_2\!:\!\sigma_1 \to F\sigma_2, \zeta\!:\!\sigma_2 \to \underline{\tau} \vdash \\ \mathsf{let}\, x_1\, \mathsf{be}\, \zeta_1\, \mathsf{in}(\mathsf{let}\, x_2\, \mathsf{be}\, \zeta_2 x_1\, \mathsf{in}\, \zeta x_2) \\ = \mathsf{let}\, x_2\, \mathsf{be}\, (\mathsf{let}\, x_1\, \mathsf{be}\, \zeta_1\, \mathsf{in}\, \zeta_2 x_1)\, \mathsf{in}\, \zeta x_2 \end{aligned}$$

are derivable.

The proof uses the induction principle instead of the structural induction used in Theorem 4. Structural induction is not only unwieldy due to the large number of term constructors, but also fails to prove the theorem in the presence of effect variables. In a similar way, we can prove a non-schematic version of Proposition 5.

For each operation $op : \vec{\beta}; \vec{\alpha}_1, \ldots, \vec{\alpha}_n$, we can define its *generic effect*

$$\mathsf{gen}_{op} : \prod \vec{\beta} \to F(\prod \vec{\alpha}_1 + \cdots + \prod \vec{\alpha}_n)$$

by

$$\mathsf{gen}_{op} \equiv_{\mathrm{def}} \lambda y : \prod \vec{\beta}.op_{\overrightarrow{\mathsf{prj}_y}}(\overrightarrow{(\vec{x} : \vec{\alpha}).\, \mathsf{inj}_{\mathsf{return}}\langle \vec{x} \rangle}) \ ,$$

using evident abbreviations, in particular, $\prod \vec{\alpha}$ stands for $\alpha_1 \times \cdots \times \alpha_m$ (see [21] for a discussion of operations and generic effects). An example is $\mathsf{gen}_{\mathsf{lookup}} : \mathbf{loc} \to F\mathbf{dat}$, which applied to a location l returns the datum stored there, and is usually written as $!l$.

Operations are recoverable from their generic effects. For example, if the operation is of the form $op : \vec{\beta}; \vec{\alpha}$, we have

$$\Gamma; \Delta \vdash op_{\vec{v}}((\vec{x} : \vec{\alpha}).t) = \mathsf{let}\, y\, \mathsf{be}\, \mathsf{gen}_{op}\langle \vec{v} \rangle\, \mathsf{in}\, t[\overrightarrow{\mathsf{prj}_y}/\vec{x}] \ ,$$

while in the general case, we use pattern matching.

Generic effects are often used in programming, as in the example above, but are not useful for logic, as the equations of the effect theory are written using the operations.

3.3 Modalities

We define local modalities in order to reason about computations. A *pureness modality* expresses the properties of a computation in terms of the returned values, while an *operation modality* expresses its properties in terms of its immediate subcomputations. Because of the exogenous view, modalities are operators on predicates, rather than propositions.

We define pureness modalities $[\downarrow]$ and $\langle\downarrow\rangle$ for a predicate $\pi : (\sigma) \to \mathbf{prop}$, and operation modalities $[op]$ and $\langle op \rangle$ for an operation $op : \vec{\beta}; \vec{\alpha}_1, \ldots, \vec{\alpha}_n$ and a predicate $\pi : (\vec{\beta}; \prod \vec{\alpha}_1 \to \underline{\tau}, \ldots, \prod \vec{\alpha}_n \to \underline{\tau}) \to \mathbf{prop}$ by

$$\begin{aligned}
[\downarrow](\pi) &\equiv_{\mathrm{def}} (\zeta : F\sigma).\forall x : \sigma.\zeta = \mathsf{return}\, x \Rightarrow \pi(x) \\
\langle\downarrow\rangle(\pi) &\equiv_{\mathrm{def}} (\zeta : F\sigma).\exists x : \sigma.\zeta = \mathsf{return}\, x \wedge \pi(x) \\
[op](\pi) &\equiv_{\mathrm{def}} (\zeta : \underline{\tau}).\forall \vec{y} : \vec{\beta}, \vec{\zeta}' : \overrightarrow{\textstyle\prod \vec{\alpha} \to \underline{\tau}}. \\
&\qquad \zeta = op_{\vec{y}}(\overrightarrow{(\vec{x} : \vec{\alpha}).\zeta'\langle \vec{x} \rangle}) \Rightarrow \pi(\vec{y}, \vec{\zeta}') \\
\langle op \rangle(\pi) &\equiv_{\mathrm{def}} (\zeta : \underline{\tau}).\exists \vec{y} : \vec{\beta}, \vec{\zeta}' : \overrightarrow{\textstyle\prod \vec{\alpha} \to \underline{\tau}}. \\
&\qquad \zeta = op_{\vec{y}}(\overrightarrow{(\vec{x} : \vec{\alpha}).\zeta'\langle \vec{x} \rangle}) \wedge \pi(\vec{y}, \vec{\zeta}')
\end{aligned}$$

The notation for the pureness modality follows the notation for Moggi's pureness predicate $t\downarrow$, which is expressible in terms of the pureness modality as $\langle\downarrow\rangle((x : \sigma).\top)(t)$.

We define $[-](\pi)$ for a predicate $\pi : (\underline{\tau}) \to \mathbf{prop}$ to be

$$(\zeta : \underline{\tau}). \bigwedge_{op : \vec{\beta}; \vec{\alpha}_1, \ldots, \vec{\alpha}_n \in \Sigma_{\mathrm{op}}} [op]((\vec{y}, \vec{\zeta}).\bigwedge_{i=1}^{n} \forall \vec{x}_i : \vec{\alpha}_i.\pi(\zeta_i\langle \vec{x}_i \rangle)))(\zeta) \ .$$

and $\langle - \rangle(\pi)$ is defined dually. Intuitively, $[-](\pi)(t)$ states that all immediate subcomputations of t satisfy π.

The derived introduction/elimination rules for necessity modalities are

$$\frac{\Gamma; \Delta, \zeta : F\sigma; \Pi \mid \Psi \vdash [\downarrow](\pi)(\zeta)}{\Gamma, x : \sigma; \Delta; \Pi \mid \Psi[\mathsf{return}\, x/\zeta] \vdash \pi(x)}$$

and

$$\frac{\Gamma; \Delta, \zeta : \underline{\tau}; \Pi \mid \Psi \vdash [op](\pi)(\zeta)}{\Delta, \vec{y} : \vec{\beta}, \vec{\zeta}' : \overrightarrow{\textstyle\prod \vec{\alpha} \to \underline{\tau}} \mid \Psi[op_{\vec{y}}(\overrightarrow{(\vec{x} : \vec{\alpha}).\zeta'\langle \vec{x} \rangle})/\zeta] \vdash \pi(\vec{y}, \vec{\zeta}')}$$

and dually for the possibility modalities. From the adjoint form of those rules, one can see that in the categorical approach to logic, pureness and operation modalities are quantifiers corresponding to the inclusion of value terms into computation terms and to operations, respectively.

To extend local to global reasoning, we use predicate fixed points to define a global necessity modality $\Box\pi$ by $\nu X : (\underline{\tau}).\pi \wedge [-](X)$ and a global possibility modality $\Diamond\pi$ by $\mu X : (\underline{\tau}).\pi \vee \langle - \rangle(X)$. In the same way, we can define other global modalities known from computational tree logic, such as AF or EG, although one should recall that as we are working in Set, all computations are finite.

Intuitively, $\Gamma \vdash (\Box\pi)(t)$ states that all subcomputations of t after some effects satisfy π. Since the subcomputation relation is reflexive and transitive, we expect the global modalities to satisfy the $S4$ axioms.

Proposition 10. *The rules*

$$\frac{\Gamma; \Delta, \zeta : \underline{\tau}; \Pi \mid \vdash \pi(\zeta)}{\Gamma; \Delta, \zeta : \underline{\tau}; \Pi \mid \vdash (\Box\pi)(\zeta)}$$

$$\begin{aligned}
&\Gamma; \Delta, \zeta : \underline{\tau}; \Pi \mid (\Box(\pi_1 \Rightarrow \pi_2))(\zeta) \vdash (\Box\pi_1 \Rightarrow \Box\pi_2)(\zeta) \\
&\Gamma; \Delta, \zeta : \underline{\tau}; \Pi \mid (\Box\pi)(\zeta) \vdash \pi(\zeta) \\
&\Gamma; \Delta, \zeta : \underline{\tau}; \Pi \mid (\Box\pi)(\zeta) \vdash (\Box\Box\pi)(\zeta)
\end{aligned}$$

together with the dual ones for $\Diamond$, are derivable.

3.4 Semantics

We start with an interpretation $\mathfrak{I}$, determined by sets $[\![\alpha]\!]$ for each base type α, functions $[\![f]\!] : [\![\vec{\alpha}]\!] \to [\![\beta]\!]$ for each

base function $f : (\vec{\alpha}) \to \beta$, and subsets $[\![R]\!]$ of $[\![\vec{\alpha}]\!]$ for each base relation $R : (\vec{\alpha}) \to \mathbf{form}$. This determines the interpretation of the rest of the logic. We interpret value formulae $\Gamma \vdash \varphi : \mathbf{form}$ in the standard way using subsets and, again, consider only interpretations that are sound with respect to the value theory $\mathfrak{V}$.

Although the effect theory $\mathfrak{E}$ is not an equational theory, it is an abbreviation for an infinitary one, given a fixed interpretation $\mathfrak{I}$ where all the base types occurring in the arity types of operations are interpreted by countable sets. In this case, which we assume from now on, the effect theory gives rise to a countable Lawvere theory L and adjoint functors $F \dashv U \colon \mathrm{Mod}_L(\mathsf{Set}) \to \mathsf{Set}$ in a standard way [26]. We are only interested in interpretations $\mathfrak{I}$ such that L is non-trivial.

Value types σ are interpreted by sets $[\![\sigma]\!]$, while computation types $\underline{\tau}$ are interpreted by models $[\![\underline{\tau}]\!]$ of the theory L. The value types are interpreted by $[\![U\underline{\tau}]\!] = U[\![\underline{\tau}]\!]$ and in the obvious way in other cases, while computation types are interpreted by

$$[\![F\sigma]\!] = F[\![\sigma]\!] \qquad [\![\mathbf{1}]\!] = \mathbf{1}$$
$$[\![\underline{\tau}_1 \times \underline{\tau}_2]\!] = [\![\underline{\tau}_1]\!] \times [\![\underline{\tau}_2]\!] \qquad [\![\sigma \to \underline{\tau}]\!] = [\![\underline{\tau}]\!]^{[\![\sigma]\!]} \ ,$$

where the model structure is defined component-wise for $M_1 \times M_2$ and point-wise for M^A.

The context $x_1 : \sigma_1, \ldots, x_n : \sigma_n$ is interpreted by $[\![\vec{\sigma}]\!] = [\![\sigma_1]\!] \times \cdots \times [\![\sigma_n]\!]$, while $\zeta_1 : \underline{\tau}_1, \ldots, \zeta_n : \underline{\tau}_n$ is interpreted by $U[\![\vec{\underline{\tau}}]\!] = U[\![\underline{\tau}_1]\!] \times \cdots \times U[\![\underline{\tau}_n]\!]$.

Value terms $\Gamma; \Delta \vdash v : \sigma$ are interpreted by functions $[\![v]\!] \colon [\![\Gamma]\!] \times [\![\Delta]\!] \to [\![\sigma]\!]$, and computation terms $\Gamma; \Delta \vdash t : \underline{\tau}$ are interpreted by functions $[\![t]\!] \colon [\![\Gamma]\!] \times [\![\Delta]\!] \to U[\![\underline{\tau}]\!]$, all defined in a straightforward way.

Note that computation terms can be interpreted as morphisms in the co-Kleisli category of the adjunction between F and U. They are of the form $A \times UM \to UN$, where $A = \prod_i [\![\sigma_i]\!]$ and $UM = U \prod_j [\![\underline{\tau}_j]\!] = \prod_j U[\![\underline{\tau}_j]\!]$. The interpretation is then equal to one of the form $UM \to U(N^A)$ and furthermore to one of the form $FUM \to N^A$, which is a morphism in the co-Kleisli category.

Contexts

$$\Pi = X_1 : (\vec{\sigma}_1; \vec{\underline{\tau}}_1) \to \mathbf{prop}, \ldots, X_n : (\vec{\sigma}_n; \vec{\underline{\tau}}_n) \to \mathbf{prop}$$

are interpreted by sets

$$[\![\Pi]\!] = \mathcal{P}([\![\vec{\sigma}_1]\!] \times U[\![\vec{\underline{\tau}}_1]\!]) \times \cdots \times \mathcal{P}([\![\vec{\sigma}_n]\!] \times U[\![\vec{\underline{\tau}}_n]\!]) \ ,$$

propositions $\Gamma; \Delta; \Pi \mid \varphi : \mathbf{prop}$ by subsets

$$[\![\varphi]\!] \subseteq [\![\Gamma]\!] \times [\![\Delta]\!] \times [\![\Pi]\!] \ ,$$

and predicates $\Gamma; \Delta; \Pi \mid \pi : (\vec{\sigma}; \vec{\underline{\tau}}) \to \mathbf{prop}$ by maps

$$[\![\pi]\!] \colon [\![\Gamma]\!] \times [\![\Delta]\!] \times [\![\Pi]\!] \to \mathcal{P}([\![\vec{\sigma}]\!] \times U[\![\vec{\underline{\tau}}]\!]) \ ,$$

all defined in an obvious way. In particular, fixed points are defined as follows: the interpretation of a predicate $\Gamma; \Delta; \Pi, X : (\vec{\sigma}; \vec{\underline{\tau}}) \to \mathbf{prop} \vdash \pi : (\vec{\sigma}; \vec{\underline{\tau}}) \to \mathbf{prop}$ defines a monotone operator $[\![\pi]\!]_a$ on $\mathcal{P}([\![\vec{\sigma}]\!] \times U[\![\vec{\underline{\tau}}]\!])$ for each $a \in [\![\Gamma]\!] \times [\![\Delta]\!] \times [\![\Pi]\!]$. By Tarski's fixed point theorem, $[\![\pi]\!]_a$ has a smallest fixed point S_a, and we define $[\![\mu X : (\vec{\sigma}; \vec{\underline{\tau}}).\pi]\!]$ to be the map $a \mapsto S_a$.

A judgement $\Gamma; \Delta; \Pi \mid \varphi_1, \ldots, \varphi_n \vdash \varphi$ is sound with respect to the interpretation $\mathfrak{I}$, if $\bigcap_{i=1}^n [\![\varphi_i]\!] \subseteq [\![\varphi]\!]$. Showing the soundness of the reasoning rules is straightforward: the structural rules and rules for connectives and quantifiers are the standard ones, the proof of soundness of equations is straightforward, the interpretation of fixed points is sound by definition, and the proofs of the soundness of the induction and free algebra principles proceed using the universal property of the free model. If L is non-trivial, the following *consistency* proposition holds

$$\forall x_1, x_2 : \sigma.\ \mathsf{return}\, x_1 = \mathsf{return}\, x_2 \Rightarrow x_1 = x_2 \ .$$

4 Embracing other approaches

4.1 Computational λ-calculus

The computational λ-calculus [14] has a pureness predicate $\Gamma \vdash_{\lambda_c} t{\downarrow}$, which states that a computation term t causes no effects, in place of the separation between values and computations. The base functions of the computational λ-calculus can be of an arbitrary type and can cause arbitrary effects. Since the main premise of our approach is that algebraic operations are an adequate representation of effects, we argue that instead of arbitrary primitive functions, we need only pure functions $f : \prod \vec{\alpha} \to \beta$ and generic effects $\mathsf{gen}_{op} : \prod \vec{\beta} \to F(\prod \vec{\alpha})$ for each operation $op : \vec{\beta}; \vec{\alpha}$ (for more general generics, one would add sum types to Moggi's language). Under this mild assumption, we get an embrace of the computational λ-calculus by translating types as

$$\alpha^{\triangleright} = \alpha \qquad (\sigma_1 \times \sigma_2)^{\triangleright} = \sigma_1^{\triangleright} \times \sigma_2^{\triangleright}$$
$$\mathbf{1}^{\triangleright} = \mathbf{1} \qquad (\sigma \to \sigma')^{\triangleright} = U(\sigma^{\triangleright} \to F\sigma'^{\triangleright})$$
$$(T\sigma)^{\triangleright} = UF\sigma^{\triangleright} \ ,$$

contexts $\Gamma = x_1 : \sigma_1, \ldots, x_n : \sigma_n$ as

$$\Gamma^{\triangleright} = x_1 : \sigma_1^{\triangleright}, \ldots, x_n : \sigma_n^{\triangleright} \ ,$$

terms as

$$x^{\triangleright} = \mathsf{return}\, x$$
$$f(t)^{\triangleright} = \mathsf{let}\, x\, \mathsf{be}\, t^{\triangleright}\, \mathsf{in}\, \mathsf{return}\, f(x)$$
$$\mathsf{gen}_{op}(t)^{\triangleright} = \mathsf{let}\, x\, \mathsf{be}\, t^{\triangleright}\, \mathsf{in}\, \mathsf{gen}_{op}\, x$$
$$[t]^{\triangleright} = \mathsf{return}\, \mathsf{thunk}\, t$$
$$\mu(t)^{\triangleright} = \mathsf{let}\, x\, \mathsf{be}\, t\, \mathsf{in}\, \mathsf{force}\, x$$

$$
\begin{aligned}
\star^{\triangleright} &= \mathsf{return}\,\star \\
\langle t_1, t_2\rangle^{\triangleright} &= \mathsf{let}\,x_1\,\mathsf{be}\,t_1\,\mathsf{in}\,\mathsf{let}\,x_2\,\mathsf{be}\,t_2\,\mathsf{in}\,\mathsf{return}\langle x_1, x_2\rangle \\
(\mathsf{fst}\,t)^{\triangleright} &= \mathsf{let}\,x\,\mathsf{be}\,t\,\mathsf{in}\,\mathsf{return}\,\mathsf{fst}\,x \\
(\mathsf{snd}\,t)^{\triangleright} &= \mathsf{let}\,x\,\mathsf{be}\,t\,\mathsf{in}\,\mathsf{return}\,\mathsf{snd}\,x \\
(\mathsf{let}\,x\,\mathsf{be}\,t\,\mathsf{in}\,t')^{\triangleright} &= \mathsf{let}\,x\,\mathsf{be}\,t^{\triangleright}\,\mathsf{in}\,t'^{\triangleright} \\
(\lambda x{:}\sigma.t)^{\triangleright} &= \mathsf{return}\,\mathsf{thunk}\,\lambda x{:}\sigma^{\triangleright}.t^{\triangleright} \\
(tt')^{\triangleright} &= \mathsf{let}\,x\,\mathsf{be}\,t^{\triangleright}\,\mathsf{in}\,\mathsf{let}\,y\,\mathsf{be}\,t'^{\triangleright}\,\mathsf{in}\,(\mathsf{force}\,x)y\ ,
\end{aligned}
$$

and judgements as

$$
\begin{aligned}
(\Gamma \vdash_{\lambda_c} t{:}\sigma)^{\triangleright} &= (\Gamma^{\triangleright} \vdash t^{\triangleright} : F\sigma^{\triangleright}) \\
(\Gamma \vdash_{\lambda_c} t_1 = t_2)^{\triangleright} &= (\Gamma^{\triangleright} \vdash t_1^{\triangleright} = t_2^{\triangleright}) \\
(\Gamma \vdash_{\lambda_c} t \downarrow \sigma)^{\triangleright} &= (\Gamma^{\triangleright} \vdash \langle\downarrow\rangle((x{:}\sigma^{\triangleright}).\top)t^{\triangleright})\ .
\end{aligned}
$$

Proposition 11. *If* $\Gamma \vdash_{\lambda_c} \varphi$ *then* $(\Gamma \vdash_{\lambda_c} \varphi)^{\triangleright}$.

4.2 Hennessy-Milner logic

Hennessy-Milner logic examines whether a given CCS process P satisfies a property φ, where processes and properties are given by

$$
\begin{aligned}
P, Q, R &::= 0 \mid a.P \mid P + Q \\
\varphi &::= \top \mid \bot \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid [a](\varphi) \mid \langle a\rangle(\varphi)\ .
\end{aligned}
$$

with a ranging over a set of actions A. Note that we deal only with finite processes. The dual properties are defined in terms of negation. Satisfiability $P \models \varphi$ and the transition relation $P \xrightarrow{a} Q$ are given in the usual way [7].

For the embrace, we take: operations $0 : 0$, $a.- : 1$ for each $a \in A$, and $+ : 2$; and equations

$$
\begin{aligned}
\xi_1 + (\xi_2 + \xi_3) &= (\xi_1 + \xi_2) + \xi_3 \qquad & \xi + \xi &= \xi \\
\xi_1 + \xi_2 &= \xi_2 + \xi_1 & \xi + 0 &= \xi\ .
\end{aligned}
$$

We then represent each process P by a computation term $\vdash P^{\triangleright} : F\mathbf{0}$ in the evident way.

Lemma 12. *The map that sends a process* P *to* $[\![P^{\triangleright}]\!]$ *induces a bijection between equivalence classes of bisimilar processes and elements of the free model* $[\![F\mathbf{0}]\!]$.

Proposition 13. *For processes* P *and* Q*, we have* $P \xrightarrow{a} Q$ *if and only if there exists a process* R *such that*

$$\vdash P^{\triangleright} = (a.Q + R)^{\triangleright}\ .$$

Proof. First, let us assume that $P \xrightarrow{a} Q$ and proceed by induction on P. If $P = 0$, then $P \xrightarrow{a} Q$ does not hold for any Q. If $P = a.P'$ and $P \xrightarrow{a} Q$, we have $P' = Q$ and $P^{\triangleright} = (a.Q + 0)^{\triangleright}$. If $P = P_1 + P_2$, then either $P_1 \xrightarrow{a} Q$ or $P_2 \xrightarrow{a} Q$. In the first case, we have $P^{\triangleright} = (a.Q + (P_2 + R))^{\triangleright}$ where $P_1^{\triangleright} = (a.Q + R)^{\triangleright}$. In the second case we proceed in the same way.

If we assume $\vdash P^{\triangleright} = (a.Q + R)^{\triangleright}$ for some R, we get $P \simeq a.Q + R$ by the soundness of the interpretation in the free model and Lemma 12. Since $a.Q + R \xrightarrow{a} Q$, we get $P \xrightarrow{a} Q$. □

We define the translation of formulae into predicates by

$$
\begin{aligned}
(\varphi_1 \wedge \varphi_2)^{\triangleright} &= (\zeta{:}F\mathbf{0}).\varphi_1^{\triangleright}(\zeta) \wedge \varphi_2^{\triangleright}(\zeta) \\
([a](\varphi))^{\triangleright} &= [+]([a.-](\varphi^{\triangleright}), (\zeta{:}F\mathbf{0}).\top) \\
(\langle a\rangle(\varphi))^{\triangleright} &= \langle +\rangle(\langle a.-\rangle(\varphi^{\triangleright}), (\zeta{:}F\mathbf{0}).\bot)\ .
\end{aligned}
$$

and similarly in other cases.

With that translation, we get a strong embrace of Hennessy-Milner logic. This shows how to express the modalities of Hennessy-Milner logic in terms of the local modalities given by the operations; we conjecture that the converse fails: that, in a suitable sense, the operation modalities cannot be expressed by the modalities of Hennessy-Milner logic.

Lemma 14. *For any process* $P = \sum a_i.P_i$ *and action* a *we have*

$$\zeta{:}F\mathbf{0}, \zeta'{:}F\mathbf{0} \mid \textstyle\sum a_i.P_i^{\triangleright} = a.\zeta + \zeta' \vdash \bigvee_{a_i = a} \zeta = P_i^{\triangleright}\ .$$

Proof. The proof employs the free algebra principle using an algebra defined on a sum of $\mathbf{1}$'s, which we regard as the set of all bisimulation equivalence classes $[Q_1 + \cdots + Q_n]$, where each Q_j is either a subterm of P or $w.0$, for some action w not occurring in P.

This has an evident semi-lattice with a zero structure, and we define $a.[Q_1 + \cdots + Q_n]$ to be $[a.(Q_1 + \cdots + Q_n)]$ if $a.(Q_1 + \cdots + Q_n)$ is a subterm of P and $[w.0]$ otherwise. □

Proposition 15. $P \models \varphi$ *holds if and only if* $\vdash \varphi^{\triangleright}(P^{\triangleright})$.

Proof. We proceed by induction on φ. The propositional cases are evident.

In the case where $P \models [a](\varphi)$ we have $P \simeq \sum a_i.P_i$ for some a_i and P_i and $P_i \models \varphi$ whenever $a = a_i$. Next, we have that $\vdash ([a](\varphi))^{\triangleright}(P^{\triangleright})$ if, and only if,

$$\zeta{:}F\mathbf{0}, \zeta'{:}F\mathbf{0} \mid P^{\triangleright} = a.\zeta + \zeta' \vdash \varphi^{\triangleright}(\zeta)$$

and so, by Lemma 14, if

$$\zeta{:}F\mathbf{0} \mid \bigvee_{a_i = a} \zeta = P_i^{\triangleright} \vdash \varphi^{\triangleright}(\zeta)\ ,$$

which holds as we know that $\vdash \varphi^{\triangleright}(P_i^{\triangleright})$ whenever $a = a_i$ by the induction hypothesis. The converse is straightforward using Proposition 13.

In the case where $P \models \langle a\rangle(\varphi)$, there exists a Q such that $P \xrightarrow{a} Q$ and $Q \models \varphi$. From the induction hypothesis, we get

$\vdash \varphi^{\triangleright}(Q^{\triangleright})$, which using the fact that $P^{\triangleright} = (a.Q + R)^{\triangleright}$ for some R implies $\vdash (\langle a\rangle(\varphi))^{\triangleright}(P^{\triangleright})$.

On the other hand, if we have $\vdash (\langle a\rangle(\varphi))^{\triangleright}(P^{\triangleright})$, we get $\vdash \exists\zeta, \zeta'.P^{\triangleright} = a.\zeta + \zeta' \wedge \varphi^{\triangleright}(\zeta)$ and from the soundness of interpretation, we show the implication in the other direction. □

Corollary 16. *Processes P and Q are bisimilar if and only if $\vdash P^{\triangleright} = Q^{\triangleright}$.*

Proof. From $\vdash P^{\triangleright} = Q^{\triangleright}$, it follows by congruence that $\vdash \varphi^{\triangleright}(P^{\triangleright})$ if and only $\vdash \varphi^{\triangleright}(Q^{\triangleright})$, and hence $P \models \varphi$ if and only if $Q \models \varphi$ for all properties φ. Since Hennessy-Milner logic classifies bisimilar processes [7], we get that $P \simeq Q$. On the other hand, bisimilarity is characterised by the four equations of our effect theory $\mathfrak{E}$, hence $P \simeq Q$ implies $\vdash P^{\triangleright} = Q^{\triangleright}$. □

4.3 Evaluation logic

Evaluation logic [17, 15, 16] reasons about computations in terms of values they return. The necessity modality $[\mathsf{let}\, x\, \mathsf{be}\, t](\pi)$ states that every value computed by computation term t satisfies φ. For example, if the effect at hand is nondeterminism, then $[\mathsf{let}\, x\, \mathsf{be}\, t](\varphi)$ holds if and only if all values computed by t satisfy φ; if it is exceptions, then $[\mathsf{let}\, x\, \mathsf{be}\, t](\varphi)$ holds if and only if t satisfies φ when it does not raise an exception. The possibility modality is defined dually: $\langle \mathsf{let}\, x\, \mathsf{be}\, t\rangle(\varphi)$ states that there exists a value computed by computation term t that satisfies φ.

We translate types of the evaluation logic by

$$\alpha^{\triangleright} = \alpha \qquad (T\sigma)^{\triangleright} = UF\sigma^{\triangleright} \ ,$$

terms by

$$\begin{aligned} x^{\triangleright} &= x \\ [t]^{\triangleright} &= \mathsf{thunk}\,\mathsf{return}\, t \\ (\mathsf{let}\, x\, \mathsf{be}\, t\, \mathsf{in}\, t')^{\triangleright} &= \mathsf{thunk}\,\mathsf{let}\, x\, \mathsf{be}\,\mathsf{force}\, t^{\triangleright}\, \mathsf{in}\,\mathsf{force}\, t'^{\triangleright} \ , \end{aligned}$$

contexts by

$$(x_1 : \sigma_1, \ldots, x_n : \sigma_n)^{\triangleright} = x_1 : \sigma_1^{\triangleright}, \ldots, x_n : \sigma_n^{\triangleright} \ ,$$

and formulae by

$$\begin{aligned} (t_1 = t_2)^{\triangleright} &= (t_1^{\triangleright} = t_2^{\triangleright}) \\ \bot^{\triangleright} &= \bot \\ (\varphi_1 \vee \varphi_2)^{\triangleright} &= \varphi_1^{\triangleright} \vee \varphi_2^{\triangleright} \\ ([\mathsf{let}\, x\, \mathsf{be}\, t](\varphi))^{\triangleright} &= \Box_{\downarrow}((x : \sigma^{\triangleright}).\varphi^{\triangleright})(t^{\triangleright}) \ , \end{aligned}$$

where

$$\Box_{\downarrow}(\pi) \equiv_{\mathrm{def}} \mu X : (F\sigma).(\zeta : F\sigma).[\downarrow](\pi)(\zeta) \vee \bigvee_{op:\vec{\beta};\vec{\alpha}_1,\ldots,\vec{\alpha}_n \in \Sigma_{\mathrm{op}}} \langle op\rangle((\vec{y},\vec{\zeta}).\bigwedge_{i=1}^{n} \forall \vec{x}_i : \vec{\alpha}_i.X(\zeta_i\langle\vec{x}_i\rangle))(\zeta) \ .$$

This agrees with Moggi's definition of the evaluation modality in Set [16].

We write $\Gamma \vdash^{M}_{ev} \varphi$ for judgements in Pitts' evaluation logic [17], but with the modality rules limited to Moggi's derived ones in [15] and their duals.

Proposition 17. *If $\Gamma \vdash^{M}_{ev} \varphi$, then $\Gamma^{\triangleright} \vdash \varphi^{\triangleright}$.*

Hoare logic [8] for finite commands and a state with locations $l_1, \ldots, l_n$ can be embraced by externalising the state [16], translating Hoare triples $\{\varphi(\vec{x})\}t\{\varphi'(\vec{x},\vec{y})\}$ to

$$\begin{aligned} [&\mathsf{let}\, \langle \vec{x}, \vec{y}\rangle\, \mathsf{be} \\ &\quad \mathsf{let}\, x_1\, \mathsf{be}\, !l_1\, \mathsf{in} \ldots \mathsf{let}\, x_n\, \mathsf{be}\, !l_n\, \mathsf{in}\, \mathsf{let}\, z\, \mathsf{be}\, t\, \mathsf{in} \\ &\quad \mathsf{let}\, y_1\, \mathsf{be}\, !l_1\, \mathsf{in} \ldots \mathsf{let}\, y_n\, \mathsf{be}\, !l_n\, \mathsf{in} \\ &\mathsf{return}\langle \vec{x}, \vec{y}\rangle](\varphi(\vec{x}) \Rightarrow \varphi'(\vec{x},\vec{y})) \ . \end{aligned}$$

However, this does not seem natural to us. The answer may lie in a coalgebraic treatment [27, 24] of state, as an algebraic treatment already failed [23] to give a natural operational semantics for state. Such a treatment could fit well with Pitts' 'ad hoc' approach to state [17].

5 Recursion

We sketch a version of Scott's LCF [30, 6], adapted to algebraic computational effects, but make no claim of definitiveness. The logic is an extension of our logic for algebraic effects over Set, based instead on the category ω-Cpo of ω-cpos and continuous maps.

We extend the value theory with inequations of the form $v_1 \leq v_2$ and suitable axioms and rules, including asymmetry. In the effect theory we use inequations, for example $\xi_1, \xi_2 \vdash \xi_1 \leq \mathsf{or}(\xi_1, \xi_2)$, and assume the existence of an operation $\Omega : 0$, and an equation $\xi \vdash \Omega() \leq \xi$ [10]. At the level of computation terms, we add recursion with

$$\frac{\Gamma; \Delta, \zeta : \underline{\tau} \vdash t : \underline{\tau}}{\Gamma; \Delta \vdash \mu\zeta : \underline{\tau}.t : \underline{\tau}} \ .$$

The logic has additional atomic propositions $v_1 \leq v_2$ and $t_1 \leq t_2$. The axioms and rules are the same as before, adapted to the presence of inequations in an obvious way, except that: the principle of induction over computations is restricted to admissible propositions; and we also have the axiom $\Gamma; \Delta; \Pi \vdash t[\mu\zeta : \underline{\tau}.t/\zeta] \leq \mu\zeta : \underline{\tau}.t$ and the principle of Scott induction

$$\Gamma; \Delta; \Pi \mid \pi(\Omega()), \forall\zeta : \underline{\tau}.\pi(\zeta) \Rightarrow \pi(t) \vdash \pi(\mu\zeta : \underline{\tau}.t) \ ,$$

also restricted to admissible propositions. The definition of admissibility is complex owing to the presence of predicates and predicate variables; we do not give it here, except to note that $\nu X : (\vec{\sigma}; \vec{\tau}).\varphi$ is admissible if φ is, under suitable assumptions on X. The other axioms and rules are as in the case of the logic for Set, adapted to the presence of inequations.

We interpret values in ω-Cpo, but still interpret all base types occurring in arities by countable sets. Then the effect theory gives rise to a countable discrete Lawvere ω-cpo theory L and an adjunction $F \dashv U \colon \mathrm{Mod}_L(\omega\text{-Cpo}) \to \omega\text{-Cpo}$ in a standard way [11].

6 Future work

We have presented some evidence of the expressiveness and strength of our logic of algebraic effects, but much clearly remains to be done. To mention one example, we expect to get an embrace of global evaluation logic [5], while we have not yet investigated the embrace of dynamic logic [29].

The question of how to account for computation deconstructors, such as exception handlers [1, 13, 21] also remains open, hence so does the question of what their logic may be. Beyond Set and ω-Cpo, and without yet looking for a logic over a general category, one could still ask for logics over categories of presheaves and sheaves, for the consideration of new names or variables [4, 20], or separation logic [28], with its additional logical connectives.

Acknowledgments

The authors would like to thank Andrej Bauer, Paul Levy, John Power, Mojca Pretnar, and Alex Simpson for their insightful comments and support.

References

[1] N. Benton and A. Kennedy. Exceptional syntax. *J. Fun. Prog.*, 11(4):395–410, 2001.

[2] F. Borceux. *Handbook of Categorical Algebra*. Cambridge University Press, 1994.

[3] C. Flanagan, A. Sabry, B. F. Duba, and M. Felleisen. The essence of compiling with continuations. In *PLDI*, pages 237–247, 1993.

[4] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Form. Asp. of Comp.*, 13:341–363, 2001.

[5] S. Goncharov, L. Schröder, and T. Mossakowski. Completeness of global evaluation logic. In *31st MFPS*, pages 447–458, 2006.

[6] M. J. C. Gordon, R. Milner, and C. P. Wadsworth. *Edinburgh LCF*. Springer, 1979.

[7] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. ACM*, 32(1):137–161, 1985.

[8] C. A. R. Hoare. An axiomatic basis for computer programming. *Comm. ACM*, 12(10):576–580, 1969.

[9] M. Hyland, P. B. Levy, G. D. Plotkin, and A. J. Power. Combining algebraic effects with continuations. *Theor. Comp. Sci.*, 375(1-3):20–40, 2007.

[10] M. Hyland, G. D. Plotkin, and A. J. Power. Combining effects: Sum and tensor. *Theor. Comp. Sci.*, 357(1-3):70–99, 2006.

[11] M. Hyland and A. J. Power. Discrete Lawvere theories and computational effects. *Theor. Comp. Sci.*, 366(1-2):144–162, 2006.

[12] P. B. Levy. Call-by-push-value: Decomposing call-by-value and call-by-name. *High. Ord. Symb. Comp.*, 19(4):377–414, 2006.

[13] P. B. Levy. Monads and adjunctions for global exceptions. *Elect. Notes Theor. Comp. Sci.*, 158:261–287, 2006.

[14] E. Moggi. Notions of computation and monads. *Inform. Comp.*, 93(1):55–92, 1991.

[15] E. Moggi. A general semantics for evaluation logic. In *9th LICS*, pages 353–362, 1994.

[16] E. Moggi. A semantics for evaluation logic. *Fund. Inform.*, 22(1/2):117–152, 1995.

[17] A. M. Pitts. Evaluation logic. In *4th HOW*, pages 162–189, 1991.

[18] G. D. Plotkin. Some varieties of equational logic. In *Essays Dedicated to Joseph A. Goguen*, pages 150–156, 2006.

[19] G. D. Plotkin and A. J. Power. Adequacy for algebraic effects. In *4th FoSSaCS*, pages 1–24, 2001.

[20] G. D. Plotkin and A. J. Power. Notions of computation determine monads. In *5th FoSSaCS*, pages 342–356, 2002.

[21] G. D. Plotkin and A. J. Power. Algebraic operations and generic effects. *Appl. Cat. Struct.*, 11(1):69–94, 2003.

[22] G. D. Plotkin and A. J. Power. Logic for computational effects: Work in progress. In *6th IWFM*, 2003.

[23] G. D. Plotkin and A. J. Power. Computational effects and operations: An overview. *Elect. Notes Theor. Comp. Sci.*, 73:149–163, 2004.

[24] G. D. Plotkin and A. J. Power. Tensors of comodels and models for operational semantics. In *24th MFPS*, 2008. To appear.

[25] A. Pnueli. The temporal logic of programs. In *18th FoCS*, pages 46–57, 1977.

[26] A. J. Power. Countable Lawvere theories and computational effects. *Elect. Notes Theor. Comp. Sci.*, 161:59–71, 2006.

[27] A. J. Power and O. Shkaravska. From comodels to coalgebras: State and arrays. *Elect. Notes Theor. Comp. Sci.*, 106:297–314, 2004.

[28] J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *17th LICS*, pages 55–74, 2002.

[29] L. Schröder and T. Mossakowski. Monad-independent dynamic logic in HasCasl. *J. Log. and Comp.*, 14(4):571–619, 2004.

[30] D. S. Scott. A type-theoretical alternative to ISWIM, CUCH, OWHY. *Theor. Comp. Sci.*, 121(1-2):411–440, 1993.